LIB MATRIX by Lucien Le Roux (2016)
Homemade C library to easily implement matrices in your program.

Contents

Thanks for using this library. The following files should be featured in the "matrix" folder. If not, you may not be able to use the library correctly.

```
matrix/
├── include
│   └── matrix.h
├── libmatrix.a
├── Makefile
└── src
    ├── cpy_mat.c
    ├── del_mat.c
    ├── disp_mat.c
    ├── id_mat.c
    ├── init_mat.c
    ├── mat_m_mat.c
    ├── mat_p_mat.c
    ├── mat_pow.c
    ├── mat_x_mat.c
    └── mat_x_n.c
```

Header files

In order for the library to work properly, you need to include the following header file:
- "matrix.h"

This header file itself includes the following headers files:
- <stdlib.h>
- <stdio.h> /* use of printf */

You shall copy the provided "matrix.h" file to your main include folder, typically doing so:
$ cp ./lib/matrix/include/matrix.h ./include/

Static library

The provided library is a static one, named libmatrix.a. To use it, do not forget to link it while compiling with the -L and -lmatrix flags.

Source files and Makefile

You can at any time modify the sources however you like. You will then have to run a "make" command to create the library again. You may also want to run a "make clean" to remove object files.

Using the library

Structure and macros

The library defines the t_matrix structure:

```
typedef struct  s_matrix
{
 double      *mat;
 int         dim[2];
}            t_matrix;
```

The actual values within the matrix are stored in the mat variable.
The dim variable defines the size of the matrix, dim[0] being its height, and dim[1] its width.
Hence the two following defines for ease of use:

```
#define MAT_H dim[0]
#define MAT_W dim[1]
```

Functions

The library features 10 main functions:

```
t_matrix      init_mat(int height, int width, char** values);
t_matrix      id_mat(int dim);
t_matrix      mat_p_mat(t_matrix *m1, t_matrix *m2);
t_matrix      mat_m_mat(t_matrix *m1, t_matrix *m2);
t_matrix      mat_x_mat(t_matrix *m1, t_matrix *m2);
t_matrix      mat_x_n(t_matrix *m, double n);
t_matrix      mat_pow(t_matrix *m, int n);
t_matrix      cpy_mat(t_matrix *m);
void         del_mat(t_matrix m);
void         disp_mat(t_matrix *m);
```

For the functions to work properly, you must respect their prototype.
Here is a quick overview of the above functions:

1. init_mat

The "init_mat" function is prototyped as follows:
t_matrix	init_mat(int height, int width, char** values);

The "init_mat" function returns a matrix of size 'height'*'width', containing the values within 'values'.

Example:
```
char	*val[] = {"1", "0", "4", "5.32", NULL};
t_matrix	mat = init_mat(2, 2, val);
```

Printing the matrix will give the following output:
```
1.00	0.00
4.00	5.32
```

Notes:
- The values array must be of type specified above. It MUST be terminated with a NULL pointer.
- If the number of values specified is fewer than the total size of the matrix, it will fill with 0s. For instance, if we want to create a matrix with the values "1", "3" and "2" of size 2*2, the omitted value will be replaced by a zero, giving a matrix like {1, 3, 2, 0}.
- If too many values are given, the exceeding ones will be ignored.

2. id_mat

The "id_mat" functions is prototyped as follows:
t_matrix	id_mat(int dim);

The "id_mat" function returns the identity matrix of size 'dim'.

Example:
```
t_matrix	i3 = id_mat(3);
```

Printing the matrix will give the following output:
```
1.00	0.00	0.00
0.00	1.00	0.00
0.00	0.00	1.00
```

3. mat_p_mat

The "mat_p_mat" function is prototyped as follows:
t_matrix	mat_p_mat(t_matrix *m1, t_matrix *m2);

The "mat_p_mat" function returns the matrix resulting of the addition of 'm1' and 'm2'.

Example:
```
char	*val[] = {"1", "2", "3", "4", "5", "6", NULL};
t_matrix	mat = init_mat(3, 3, val);
t_matrix	i3 = id_mat(3);
t_matrix	new = mat_p_mat(&mat, &i3);
```

Printing the matrix will give the following ouput:
```
2.00	2.00	3.00
4.00	6.00	6.00
0.00	0.00	1.00
```

Notes:
- The two matrices must be of the same dimensions. If they are not, the function will notice it, print an error message and exit the program with status 84.

4. mat_m_mat

The "mat_m_mat" function is prototyped as follows:
t_matrix	mat_m_mat(t_matrix *m1, t_matrix *m2);

The "mat_m_mat" function returns the matrix resulting of the subtraction of 'm1' and 'm2'.

Example:
```
char	*val[] = {"1", "2", "3", "4", "5", "6", NULL};
t_matrix	mat = init_mat(3, 3, val);
t_matrix	i3 = id_mat(3);
t_matrix	new = mat_m_mat(&mat, &i3);
```

Printing the matrix will give the following ouput:
```
0.00	2.00	3.00
4.00	4.00	6.00
0.00	0.00	-1.00
```

Notes:
- The two matrices must be of the same dimensions. If they are not, the function will notice it, print an error message and exit the program with status 84.

5. mat_x_mat

The "mat_x_mat" function is prototyped as follows:
t_matrix	mat_x_mat(t_matrix *m1, t_matrix *m2);

The "mat_x_mat" function returns the matrix resulting of the multiplication of 'm1' and 'm2'.

Example:
```
char	*val1[] = {"1", "2", "3", "4", NULL};
t_matrix	mat1 = init_mat(2, 2, val1);
char	*val2[] = {"5", "6", "7", "8", NULL};
t_matrix	mat2 = init_mat(2, 2, val2);
t_matrix	new = mat_x_mat(&mat1, &mat2);
```

Printing the matrix will give the following ouput:
19.00	22.00
43.00	50.00

Notes:
- The two matrices' dimensions must be compatible for multiplication. If they are not, the function will notice it, print an error message and exit the program with status 84. As a reminder, A(p, n) and B(m, q) can be multiplied only if m = n. The result matrix will then be of dimensions (p, q).

6. mat_x_n

The "mat_x_n" function is prototyped as follows:
t_matrix	mat_x_n(t_matrix *m, double n);

The "mat_x_n" function returns the matrix resulting of the multiplication of 'm' and 'n'.

Example:
```
char	*val[] = {"1", "2", "3", "4", NULL};
t_matrix	mat = init_mat(2, 2, val);
t_matrix	new = mat_x_n(&mat, 4);
```

Printing the matrix will give the following ouput:
4.00	8.00
12.00	16.00

7. mat_pow

The "mat_pow" function is prototyped as follows:
t_matrix	mat_pow(t_matrix *m, int n);

The "mat_pow" function returns the matrix resulting of the exponentiation of 'm' to the power 'n'.

Example:
```
char	*val[] = {"1", "2", "3", "4", NULL};
t_matrix	mat = init_mat(2, 2, val);
t_matrix	new = mat_pow(&mat, 2);
```

Printing the matrix will give the following ouput:
7.00	10.00
15.00	22.00

8. cpy_mat

The "cpy_mat" function is prototyped as follows:
t_matrix	cpy_mat(t_matrix *m);

The "cpy_mat" function returns a copy of the matrix 'm'.

Example:
```
char	*val[] = {"1", "2", "3", "4", NULL};
t_matrix	mat = init_mat(2, 2, val);
t_matrix	new = cpy_mat(&mat);
```

Printing the matrix will give the following ouput:
1.00	2.00
3.00	4.00

9. del_mat

The "del_mat" function is prototyped as follows:
void	del_mat(t_matrix m);

The "del_mat" function deletes a matrix. It is mandatory if you want to avoid memory leaks.

Example:
```
char	*val[] = {"1", "2", "3", "4", NULL};
t_matrix	mat = init_mat(2, 2, val);
del_mat(mat);
```

10. disp_mat

The "disp_mat" function is prototyped as follows:
void	disp_mat(t_matrix *m);

The "disp_mat" function displays a matrix.
Example:
```
char	*val[] = {"1", "2", "3", "4", NULL};
t_matrix	mat = init_mat(2, 2, val);
disp_mat(mat);
```

Bug reports

Please report bugs to lucien.le-roux@epitech.eu.

Licensing

The LIBMATRIX was written by Lucien Le Roux.
This library is furnished as is and neither any reclamations nor complaints will be received.

#######